МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»

ІНЖЕНЕРНО-ТЕХНІЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА КОМП’ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

**ЛАБОРАТОРНА РОБОТА № 7**

з дисципліни «Паралельні та розподілені обчислення»

**«MPI попарний обмін повідомленнями»**

Студента 4-го курсу

спеціальності 123 –

«Комп'ютерна інженерія»

Комара Івана Валерійовича

м. Ужгород – 2023

**Варіант 8**

**Виконання завдань**

Розробити програму для знаходження максимального значення серед елементів матриці MA= MB+ MC.

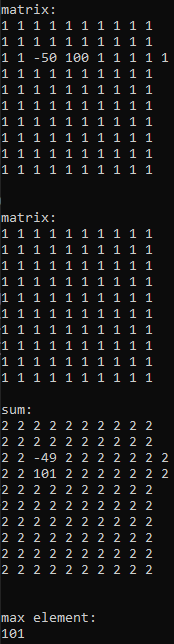


Рисунок 1 – Результат виконання програми при N = 10 і n = 4

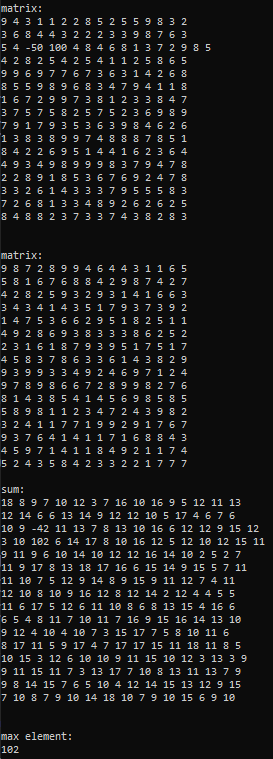


Рисунок 2 – Результат роботи програми при N = 16 і n = 8

**Висновок**: У ході виконання даної лабораторної роботи була створена програма, яка використовує блокуючі функції передачі даних MPI\_Send та MPI\_Recv для паралельного обчислення суми матриць та пошуку максимального значення в результуючій матриці. Програму було протестовано для різної кількості процесів, а отриманий результат множення було перевірено за допомогою відповідного онлайн-калькулятора.

**Лістинг програми:**

#include <mpi.h>

#include <iostream>

using namespace std;

#define MAIN\_PROCESS\_RANK 0

#define MAIN\_PROCESS\_TAG 1

#define SUB\_PROCESS\_TAG 2

const int N = 16;

MPI\_Status status;

int MB[N][N];

int MA[N][N];

int MC[N][N];

int main()

{

srand(time(NULL));

int number\_of\_processes;

int process\_rank;

int part\_size;

MPI\_Init(NULL, NULL);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &number\_of\_processes);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &process\_rank);

if (process\_rank == MAIN\_PROCESS\_RANK) {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

/\*MB[i][j] = rand() % 9 + 1;

MC[i][j] = rand() % 9 + 1;\*/

MB[i][j] = 1;

MC[i][j] = 1;

}

}

MB[2][2] = -50;

MB[2][3] = 100;

cout << endl << endl << "matrix:" << endl;

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cout << MB[i][j] << ' ';

}

cout << endl;

}

cout << endl << endl << "matrix:" << endl;

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cout << MC[i][j] << ' ';

}

cout << endl;

}

part\_size = N / number\_of\_processes;

int shift = N % number\_of\_processes;

for (int i = 0; i < N; i++) {

for (int j = 0; j < i; j++) {

int buf = MB[i][j];

MB[i][j] = MB[j][i];

MB[j][i] = buf;

buf = MC[i][j];

MC[i][j] = MC[j][i];

MC[j][i] = buf;

}

}

int\* local\_minimums = new int[number\_of\_processes];

local\_minimums[0] = INT32\_MIN;

for (int i = 0; i < part\_size + shift; i++) {

for (int j = 0; j < N; j++) {

MA[i][j] = MC[i][j] + MB[i][j];

local\_minimums[0] = local\_minimums[0] > MA[i][j] ? local\_minimums[0] : MA[i][j];

}

}

for (int i = 1; i < number\_of\_processes; i++) {

MPI\_Send(&part\_size, 1, MPI\_INT, i, MAIN\_PROCESS\_TAG, MPI\_COMM\_WORLD);

MPI\_Send(&MC[i \* part\_size + shift][0], part\_size \* N, MPI\_INT, i, MAIN\_PROCESS\_TAG, MPI\_COMM\_WORLD);

MPI\_Send(&MB[i \* part\_size + shift][0], part\_size \* N, MPI\_INT, i, MAIN\_PROCESS\_TAG, MPI\_COMM\_WORLD);

}

for (int i = 1; i < number\_of\_processes; i++) {

MPI\_Recv(&MA[i \* part\_size + shift][0], abs(part\_size \* N), MPI\_INT, i, SUB\_PROCESS\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&local\_minimums[i], 1, MPI\_INT, i, SUB\_PROCESS\_TAG, MPI\_COMM\_WORLD, &status);

}

cout << endl << "sum:" << endl;

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cout << MA[i][j] << ' ';

}

cout << endl;

}

int max = local\_minimums[0];

for (int i = 0; i < number\_of\_processes; i++) {

max = max < local\_minimums[i] ? local\_minimums[i] : max;

}

cout << endl << endl << "max element:" << endl << max << endl;

}

else {

MPI\_Recv(&part\_size, 1, MPI\_INT, MAIN\_PROCESS\_RANK, MAIN\_PROCESS\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&MC, part\_size \* N, MPI\_INT, MAIN\_PROCESS\_RANK, MAIN\_PROCESS\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&MB, part\_size \* N, MPI\_INT, MAIN\_PROCESS\_RANK, MAIN\_PROCESS\_TAG, MPI\_COMM\_WORLD, &status);

int local\_minimum = INT32\_MIN;

for (int i = 0; i < part\_size; i++) {

for (int j = 0; j < N; j++) {

MA[i][j] = MC[i][j] + MB[i][j];

local\_minimum = local\_minimum > MA[i][j] ? local\_minimum : MA[i][j];

}

}

MPI\_Send(&MA, part\_size \* N, MPI\_INT, MAIN\_PROCESS\_RANK, SUB\_PROCESS\_TAG, MPI\_COMM\_WORLD);

MPI\_Send(&local\_minimum, 1, MPI\_INT, MAIN\_PROCESS\_RANK, SUB\_PROCESS\_TAG, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}